

LINUX Expert

Sometimes it's useful to restrict access to certain parts of your website. Jon Thompson shows you how to use MySQL to create a members-only area that can be accessed only by registered users



RED HAT FOILS HACKERS

Red Hat subscribers are far less likely to fall foul of network security risks than those running other platforms, according to a report released by the SANS Institute.

The report identifies 20 of the most serious security threats. Of these, it claims, Red Hat users are susceptible to only two. Red Hat has now issued patches to close these holes.

Mark Cox, of the Red Hat security response team, said: "This report shows there are relatively few critical issues affecting users of Linux-based operating systems. However, we believe even one is unsatisfactory, and our strategy is to respond rapidly to fix these issues while innovating new technology to reduce the risk of future issues."

NOVELL OPENS LINUX GATE

On 9th August Novell launched its new openSUSE Project with the release of the beta version of SuSE 10.0. ISO images, packages and source code are available for download from www.opensuse.org and www.opensuse.de.

Novell aims to bring together a community of developers who will act as contributors and maintainers. Novell will handle packaging and marketing and pay developers for their efforts.

Greg Mancusi-Ungaro, director of marketing at SuSE Products, says the intention is to create a "lizard blizzard" of publicity, referring to SuSE's chameleon logo.

PLAYSTATION 3 IS LINUX SUPERCOMPUTER

Executive deputy president of Sony Computer Entertainment, Ken Kutagari, has claimed that his company's new PlayStation 3 is not only a games console but also a Linux supercomputer.

Sony says the Cell processor at the heart of the console delivers nearly a teraflop of computing power and will run multiple operating systems. The system will have Linux pre-installed on its hard disk, but it is unclear whether this will be included in the standard product or available as an optional extra.

Kutagari says that applications such as photo and video editing could be bundled with the Linux distribution.

This month, we're going to create a members-only area for a website using a database and a bit of web programming. In *Linux Expert*, *Shopper* July 2005 we showed you how to build a basic web server and secure access to individual pages based on a simple password file. We also hinted that a combination of the PHP programming language and the MySQL database can be used to enhance your site and even manage its security. We've included a PDF of the original article about setting up the web server in the In The Mag section of this month's cover disc.

There are many reasons to restrict access to parts of a website, but using Apache's built-in access controls can be labour-intensive. A better idea is to allow users to create their own web-based accounts. PHP can help you do this and enables you to create flexible websites powered by a database such as MySQL.

PHP is a scripting language designed to make web programming easier and it contains some very useful functions. The usage graph at www.php.net/usage.php shows how it has grown in popularity: it currently accounts for over 22 million websites on more than one million IP addresses. PHP is also secure. The server undertakes all processing and sends only the results back to the web browser, so visitors to your site never get to see the source code of your scripts.

MySQL is a relational database management system (RDBMS) and is bundled with most mainstream Linux distributions. It is a highly efficient server capable of handling more than 50 million records and has a maximum database size of eight million terabytes.

An RDBMS supports one or more databases. Each of these has one or more tables containing rows of data records. Each row contains columns of data. MySQL manages its own set of database users and these have permissions to access and change data on the databases and the tables they contain. A security system ensures that each user is only allowed to make appropriate changes to the databases, tables and records.

MySQL understands the Structured Query Language (SQL). English-like commands, called queries, are used to manipulate the database, making it comparatively easy for beginners to use. It also has a command-line interface, which we'll use to check that our PHP programs are creating data properly.

SETTING UP

Any Linux distribution that comes with Apache almost certainly includes both MySQL and PHP. Many programmers refer to this combination as LAMP (Linux, Apache, MySQL and PHP). Like Apache, there are several versions of both MySQL and PHP and we're going to use the packages

bundled with SuSE Linux Professional 9.3. This includes PHP4 version 4.3.10-14 and MySQL version 4.1.10a-3.

The easiest method of installing MySQL and PHP4 is to select their entries when you install Linux, but you can also download and install them yourself later. You can download MySQL from www.mysql.com. As root, add a user group and user for MySQL with the commands:

```
# groupadd mysql
# useradd -g mysql mysql
```

Now place the downloaded tarball in /usr/local and unpack it using:

```
# gunzip <filename> | tar -xvf -
```

Replace <filename> with the name of the downloaded file. Enter the created directory and run the configure script by typing ./configure --prefix=/usr/local/mysql. Then type the command 'make' to build the package and then 'make install' to install the software.

You can download PHP4 from www.php.net. Again in /usr/local, unpack the tarball using the above gunzip command, enter the directory this creates and configure the installation script by typing ./configure --with-apxs2=/usr/local/apache2/bin/apxs2 --with-mysql. If you installed Apache from its tarball into /usr/local, this command line should be fine. If not, the path to apxs2 will depend on your distribution, so make sure you enter the path to wherever it appears on your system.

You can now enter the 'make' and 'make install' commands as before. It's a good idea to read the accompanying Readme files before you start, just in case the developers have made any changes to the configuration parameters between releases.

You can also download the precompiled RPMs from the relevant sites and install each with the command:

```
# rpm -i <package name>
```

If you install these programs from the discs that came with your distribution using a graphical installation tool such as YaST, Apache should already understand what to do with .php scripts, but if you are building from source, you need to tell Apache what to do.

First, add php4 to the list of modules called APACHE_MODULES in the /etc/sysconfig/apache2 file. Now add the following lines to /etc/apache2/httpd.conf:

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php .php3
```



```
AddType application/x-httpd-php .php4
AddType application/x-httpd-php-source .phps
DirectoryIndex index.php
DirectoryIndex index.php3
DirectoryIndex index.php4
AddType application/x-httpd-php php4
```

However, it's far easier to install the packages straight from your distribution and let your installation tool configure them.

TEST BED

To test PHP, start your apache2 server with the command:

```
# /etc/init.d/apache2 start
```

Now create the file /srv/www/htdocs/test.php and enter the following line:

```
<?php phpinfo(); ?>
```

The /srv/www/htdocs/ directory is where Apache stores its web pages by default. Start your web browser and enter the URL localhost/test.php. You should see a web page with lots of information about your PHP installation's capabilities.

The next step is to start MySQL by typing:

```
# /etc/init.d/mysql start
```

When run for the first time, this command creates tables within a default database and warns that its root user has no password. This is not the same root account as the Linux super user root. This



The phpinfo function gives a description of the PHP environment and shows that Apache understands how to interpret the relevant sections of code

is a special account inside the MySQL system. To set the password, type mysql to start the command line interpreter, then type:

```
mysql> SET PASSWORD FOR root@localhost
=PASSWORD('<password>');
```

Substitute the new password for the string <password>. You can leave mysql by typing quit;. Don't forget to end every command with a semicolon.

MEMBERS ONLY

To protect a part of a website so that only registered and logged-in users may access it, the

first thing we need to do is create a MySQL database in which to store account details. We're going to store just the username, password, date of joining and an email address, but many sites store other things such as the user's first and last names, full postal address and whether they want to receive email notifications of site updates and other marketing information.

We'll call our database user_details, and it will contain a single table called user. Logged in as root, open a text editor and enter the following into a file called users.sql:

```
[users.sql]
CREATE DATABASE user_details;
USE user_details;
CREATE TABLE user (
    username VARCHAR(20) NOT NULL,
    password VARCHAR(20) NOT NULL,
    joindate DATE,
    email VARCHAR(30) NOT NULL,
    PRIMARY KEY (username));
INSERT INTO user (username, password,
    joindate, email)
VALUES ("guest", "nothing", "0000-00-00", "guest@cshopper.com");
```

This little script first creates our database and then commands MySQL to use this database, applying the following commands to it. It then tells SQL to create a table with four columns (username, password, joindate and email). MySQL has various data types and we're using VARCHAR to store some of the required details, which indicates a string with a variable number of characters. The number in brackets indicates the maximum number of characters each VARCHAR field can contain. This variable length means the database doesn't waste space if we use fewer characters.

To run this program, go back into MySQL's command line and enter:

```
mysql> source users.sql;
Query OK, 1 row affected (0.00 sec)
Database changed
Query OK, 0 rows affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
```

If you need to delete a database and start again during development you can use the DROP DATABASE <name>; command to delete it before running the source command again.

DO OR DIE

The next step is to create a web page that we can link to from some convenient point in the website. This will give users the opportunity to log in and go to the members-only area. It will also allow new users to create an account.

In /srv/www/htdocs, create a file called login.php and enter the login.php script that appears in the box on page 266.

This piece of HTML contains two forms. The first asks for a username and password and has a submit button that calls a PHP script that we'll write in a moment called existinglogin.php. This checks the credentials against our database, logs the user in and sends them to a page that's restricted to members only.

The second form is slightly more complex, and the associated submit button runs a PHP script called newmember.php, which creates a new

USING MYSQL FROM THE COMMAND LINE

To see the structure of the database and the data already entered by our SQL script, type the following:

```
mysql> use user_details;
Database changed
mysql> show tables from user_details;

+-----+
| Tables_in_user_details |
+-----+
| user                    |
+-----+

mysql> show columns from user;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| username | varchar(20) | | PRI | | |
| password | varchar(20) | | | | |
| joindate | date | YES | | NULL | |
| email | varchar(30) | | | | |
+-----+-----+-----+-----+-----+-----+

mysql> select * from user;
+-----+-----+-----+-----+
| username | password | joindate | email |
+-----+-----+-----+-----+
| guest | nothing | 0000-00-00 | guest@cshopper.com |
+-----+-----+-----+-----+
```

Here, we can see the guest account with the password 'nothing'. We can use this to test our website later on.

account in our MySQL database and then invites the user to log in.

Both forms, when activated, pass information using a built-in method called POST. This places the data in the fields into a special array called \$_POST. The advantage of this is that we don't have to pass information insecurely as part of the URL, which in turn means that any users who try to hack our site won't gain many free clues about the structure and any weaknesses in the underlying scripts.

When the user tries to register a new account, we want to access the values they have entered, check them and, if valid, store them in the MySQL database. In /srv/www/htdocs, create a file called newmember.php and enter the newmember.php script that appears in the box below.

In this script, substitute the MySQL root password you set up earlier for <password>. As mentioned earlier, the post method on the previous web page passes the information the user types into the form via the \$_POST array when they press the submit button marked Register me.

This script sends a query to MySQL to create the user account. The query is very much like the one in our users.sql file, except that we've used variables rather than actual values.

The script makes a connection to MySQL, which is on the same system as the web server. If the database were stored on another networked computer, we would use the IP address or DNS name instead of localhost.

Each of the MySQL PHP statements also has an or die clause. This is a convenient way of handling errors. To discover the exact error rather than outputting simple text, we can use:

```
Or die (mysql_error());
```

This is fine for development, but it could give a determined hacker some extra information they really don't need, so giving just a terse error message and ending the script is the best policy for public-facing systems.

Entering details for a new account on the login.php page and pressing the Register me! button makes the newuser.php script run and, if the credentials are OK, it will create a new record in the

THE SCRIPTS

LOGIN.PHP

```
<html>
<p>Enter your username and password to
log in:</p>
<form action="existinglogin.php"
method="POST">
  <br> Username:
  <input type="text" name="username">
  <br> Password:
  <input type="password"
name="password">
  <input type="submit" value="log
in">
</form>
<p>
<hr>
<p>Not a member? Enter your details
below for instant access:</p>
<form action="newmember.php"
method="POST">
  <br> Username:
  <input type="text"
name="newusername">
  <br> Password:
  <input type="password"
name="newpassword">
  <br> Your email:
  <input type="text" name="email">
  <input type="submit" value="Register
me!">
</form>
</html>
```

NEWMEMBER.PHP

```
<html>
<p>Creating a new username for you...
</p>
<?php
$dbhost = "localhost";
$dbname = "user_details";
$dbuser = "root";
$dbpass = "<password>";
$username = $_POST['newusername'];
$userpass = $_POST['newpassword'];
$useremail = $_POST['email'];
$today = date("Y-m-d");
if ($username == "" or $userpass
== "" or $useremail == "")
die ("<p>Blank details entered!
```

```
You must fill in all fields.");
$query = "INSERT INTO user
(username, password, joindate,
email) VALUES ('$username',
'$userpass', '$today',
'$useremail')";
$connection = mysql_connect($dbhost,
$dbuser, $dbpass)
or die ("<br>Couldn't connect to
MySQL!");
$db = mysql_select_db($dbname,
$connection)
or die ("<br>Couldn't select the
database!");
$result = mysql_query($query)
or die ("Invalid details. Maybe
this username is already
taken...");
$_SESSION['login'] = 'yes';
?>
<p>New user created. Please click <a
href="login.php"> here</a> to go to
log in.</p>
</html>
```

EXISTINGLOGIN.PHP

```
<?php
/* login */
session_start();
$dbhost = "localhost";
$dbname = "user_details";
$dbuser = "root";
$dbpass = "<password>";

$username = strip_tags(trim($_
POST['username']));
$userpass = strip_tags(trim($_
POST['password']));
$query = "SELECT password FROM user
WHERE username='$username'";

$connection = mysql_connect($dbhost,
$dbuser, $dbpass)
or die ("<br>Couldn't connect to
MySQL!");

$db = mysql_select_db($dbname,
$connection)
or die ("<br>Couldn't select the
```

```
database!");
$result = mysql_query($query)
or die ("<br>Couldn't run
query!");
$row= mysql_fetch_array
($result,MYSQL_ASSOC);
$rows = mysql_num_rows($result);
if($row['password'] != $userpass or
$rows == 0)
{
die("<br>Sorry. Invalid details
entered!");
}
mysql_close($connection);
$_SESSION['login'] = 'yes';
header ("Location: members_
area.php");
?>
```

MEMBERS_AREA.PHP

```
<?php session_start();
if ($_SESSION['login'] != 'yes')
header("Location: login.php");
?>
<html>
<p>Welcome to the members only
area!</p>
<form action="logout.php"
method="POST">
  <input type="submit" value="log
me out">
</form>
</html>
```

LOGOUT.PHP

```
<?php
session_start();
session_destroy();
unset ($_SESSION);
?>
<html>
<p>You have been logged out. Do visit
again soon.</p>
</html>
```

database. To make sure this page works, go into MySQL's command-line interpreter and enter:

```
mysql> SELECT * FROM user;
```

We can now write the piece of PHP that handles what happens when an existing user tries to log in. To do this, create a file called `existinglogin.php` and enter the existinglogin script that appears in the box opposite.

The first thing to note about this script is that we've used the `session_start()` function. This gives us convenient access to an array of global variables that we can access and manipulate on different pages. These variables are stored in the `$_SESSION` array. To add a variable to this array we just have to assign it a value, as we do at the end to set the variable `login` to the value `yes`.

The script fetches all rows from our database that match the entered username. There can be only one because we have defined the username field as the primary key. If any user attempts to create a second account with the same name, they are told that they have entered invalid details.

Next, the script checks to see if there are any matching rows (that is, that the user account exists), and checks the password to see if it is the same as the one entered. You should always give a terse error message when such credential checks fail, because pointing out that the username was fine but the password was wrong, for instance, can give hackers a head start when they're trying to crack an account.

After closing the connection with MySQL, the script creates a `$_SESSION` variable to say that the login has been successful and jumps to the members-only area using the header function.

ACCESS ALL AREAS

Now let's create `members_area.php`. Enter the `members_area.php` script from the box on the opposite page.

Again, we use `session_start()` to give access to the variables stored in the `$_SESSION` array. The first thing we do is check the login variable. If the value isn't yes, we redirect the user to the `login.php` script. This simple test is all we have to do on each restricted page to keep unregistered people, or users who have not logged in, out of such areas. Try surfing to the members' area without logging in. Your browser simply displays the login screen again.

It's useful to note that we can also use the `isset` function to check if certain variables exist within `$_SESSION`. To find out if a certain variable is set, you'd use `isset($<variable>);` in your code. To see if it has not been set use `!isset($<variable>);`. Note the use of the exclamation mark.

We need to write one final script, `logout.php`, which you'll find in the box opposite. Pressing the log me out button in the `members_area.php` script calls this to unset the login session variable.

This script uses the `session_destroy()` function to tear down the `$_SESSION` array and, for good measure, also unsets the login variable. If a user attempts to enter the members' area after this script runs, their browser will redirect them to the login screen. Interestingly, if they press the back button to try to trick the browser into letting them see the login-protected page again, this too will redirect them to the login screen.

You can also enforce good security practices with PHP. By storing the date the user last changed

REVIEW BOOK

The Linux Enterprise Cluster

RATING ★★★★★

PRICE £23

SUPPLIER www.amazon.co.uk

ISBN 1593270364

PAGES 430

Author Karl Kopper takes the reader through the process of building a Linux cluster powerful enough to service an entire organisation using a distributed, fault-tolerant computing resource that has no single point of failure. This practical guide has an accompanying CD that contains all the open-source software packages you need. But those who want to jump straight in and get to the practical aspects might find themselves frustrated.

The book has four major parts, the first two of which deal with building a two-node load-balanced server. It begins by describing the services a cluster runs and what they do. This is fairly standard and not especially cluster-oriented. Only in chapter two do we begin to get into the practical details of how a cluster handles the intercommunication between nodes. This includes a detailed description of how individual kernels route packets around the cluster in case of node failure. Part one ends with a rather light chapter on how to compile the Linux kernel.

Part two introduces the concept of high availability and begins by describing how to maintain a common system configuration between two computers using the `rsync` utility. Most of the chapter involves setting up secure communications between nodes using the secure shell (SSH). It continues with a chapter on cloning the configuration of individual systems using the secure communications channels just established. The rest of part two shows the theory and practice of monitoring the server for high availability and introduces the Heartbeat package.

Part three starts at chapter 10, and finally we get on to building a cluster. To find this so late in the book is frustrating given the title. After providing an overview of this process, the book delves into the theory of clustering and at last begins the process of building a cluster.

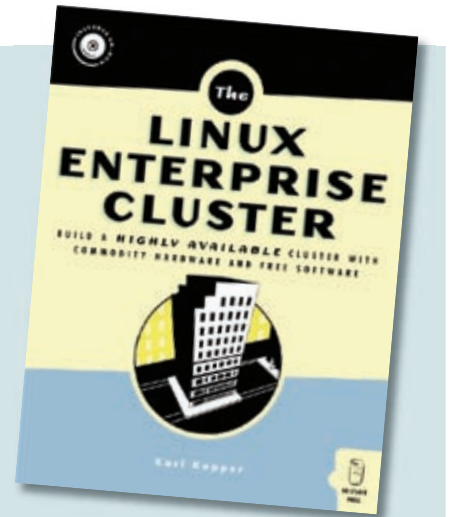
The cluster takes shape over several chapters. The techniques discussed in the first two parts

their password, you can force them to change it at regular intervals and check it for length, re-use and even whether it is the same as their username. Users sometimes forget their passwords, so you could use the email address they gave during registration to send a reminder on demand using the PHP mail function. The format of this function is:

```
mail(address, subject, message, headers)
```

The parameters are all strings.

PHP and MySQL together provide a simple and convenient mechanism for managing the creation of accounts beyond the authorisation mechanisms provided in Apache itself. Allowing the user to set up his or her own web-based accounts enables them to access the site instantly, rather than having



make the transition to a multi-node cluster, as the packages described are used to create a high-availability cluster that includes a distributed file system that can arbitrate between nodes trying to access the same resources.

The final part of the book describes health and performance monitoring and maintenance. For this, it describes both `SNMP`, which we looked at in *Linux Expert*, *Shopper* August 2005, and the sophisticated `Ganglia` package. This enables you to watch the cluster in action as it distributes its load across nodes and can create detailed usage graphs and reports. The final chapters give case studies and recap the book's content.

The inclusion of appendices on topics as basic as adding network cards and downloading packages from the net is surprising, given the level of expertise needed to build an enterprise Linux cluster. Others provide information of a more technical nature, such as using `tcpdump` and a discussion of alternative cluster file systems.

The Linux Enterprise Cluster is both complete and frustrating and could have been organised in a more straightforward way. Some chapters are rather short and the level of detail feels uneven. If you read it all and understanding the author's thought process, though, you can indeed build an enterprise-class high-availability Linux cluster.

✓ Complete, detailed and with all the software you need on an accompanying CD-ROM

✗ Not the clearest layout for a practical instruction guide

to wait for an administrator to add them to the system by hand.

By collecting relevant information from the user, it's possible to process PayPal and credit card transactions using pre-written scripts, and many sites provide free examples. You can find a good selection at www.hotscripts.com/php/scripts_and_programs/e-commerce/index.html, along with shopping carts, auction systems and online catalogue systems, all of which have been written using PHP and MySQL. 

NEXT MONTH

SYSTEM ADMINISTRATION

We show you how the free Webmin web-based Linux management console can make the Linux administrator's life easier